# PROGRAMMING NOTES

A compiler is a complex computer program that takes another program writ­ten in a high-level language and translates it into an equivalent program in the machine language of some computer. The high-level program is called source code, and the re­sulting machine code is a program that the computer can directly execute. The dashed line in the diagram represents the execution of the machine code (also known as "running the program").

A diagram of a machine code

Description automatically generated

An interpreter is a program that simulates a computer that understands a high-level language. Rather than translating the source program into a machine language equivalent, the interpreter analyses and executes the source code instruction by instruction as necessary. A diagram of a computer running an interpreter

Description automatically generated

The difference between interpreting and compiling is that compiling is a one­ shot translation; once a program is compiled, it may be run repeatedly without further need for the compiler or the source code. In the interpreted case, the interpreter and the source are needed every time the program runs.

* Parentheses must be included after the function name whenever we want to execute a function.

One important thing to be aware of is that some identifiers are part of Python itself. These names are called reserved words or keywords and cannot be used as ordinary identifiers.

A white sheet with black text

Description automatically generatedA close up of words

Description automatically generated

A diagram of a algorithm

Description automatically generated

A table with text on it

Description automatically generated

In Python, float, int, and long are built-in data types that represent numbers with different precision and size.

**int** represents integers (whole numbers) with no decimal point, and has a fixed size depending on the system architecture. In Python 3.x, the int type has unlimited precision, meaning it can represent arbitrarily large numbers.

**float** represents floating-point numbers with decimal points, and has limited precision due to the way floating-point numbers are stored in memory. Floating-point numbers are represented as binary fractions, which can cause some values to be approximated rather than represented exactly.

**long** is not a separate data type in Python 3.x, as int can represent arbitrarily large integers. However, in Python 2.x, long is a separate data type that represents integers of arbitrary size, while int is limited to the size of the system architecture.

Here is an example code snippet that demonstrates the difference between these data types:

1. a = 10
2. b = 3.14
3. c = 1234567890123456789012345678901234567890
5. print(type(a)) # int
6. print(type(b)) # float
7. print(type(c)) # int in Python 3.x, long in Python 2.x
9. d = float(a) # convert integer to float
10. e = int(b) # convert float to integer
11. f = int(c) # convert long to integer (in Python 2.x)
12. g = a + b # integer + float = float
13. h = b \* c # float \* integer = float
14. i = c / a # integer / integer = float
16. print(d, e, f, g, h, i) # 10.0, 3, 1234567890123456789012345678901234567890, 13.14, 3.8842658190123e+38, 1.2345678901234568e+32

In summary, int is used for representing integers, float for representing floating-point numbers, and long is not used in Python 3.x, as int can represent arbitrarily large integers. Converting between these data types can be done using the float(), int(), and long() functions as necessary.